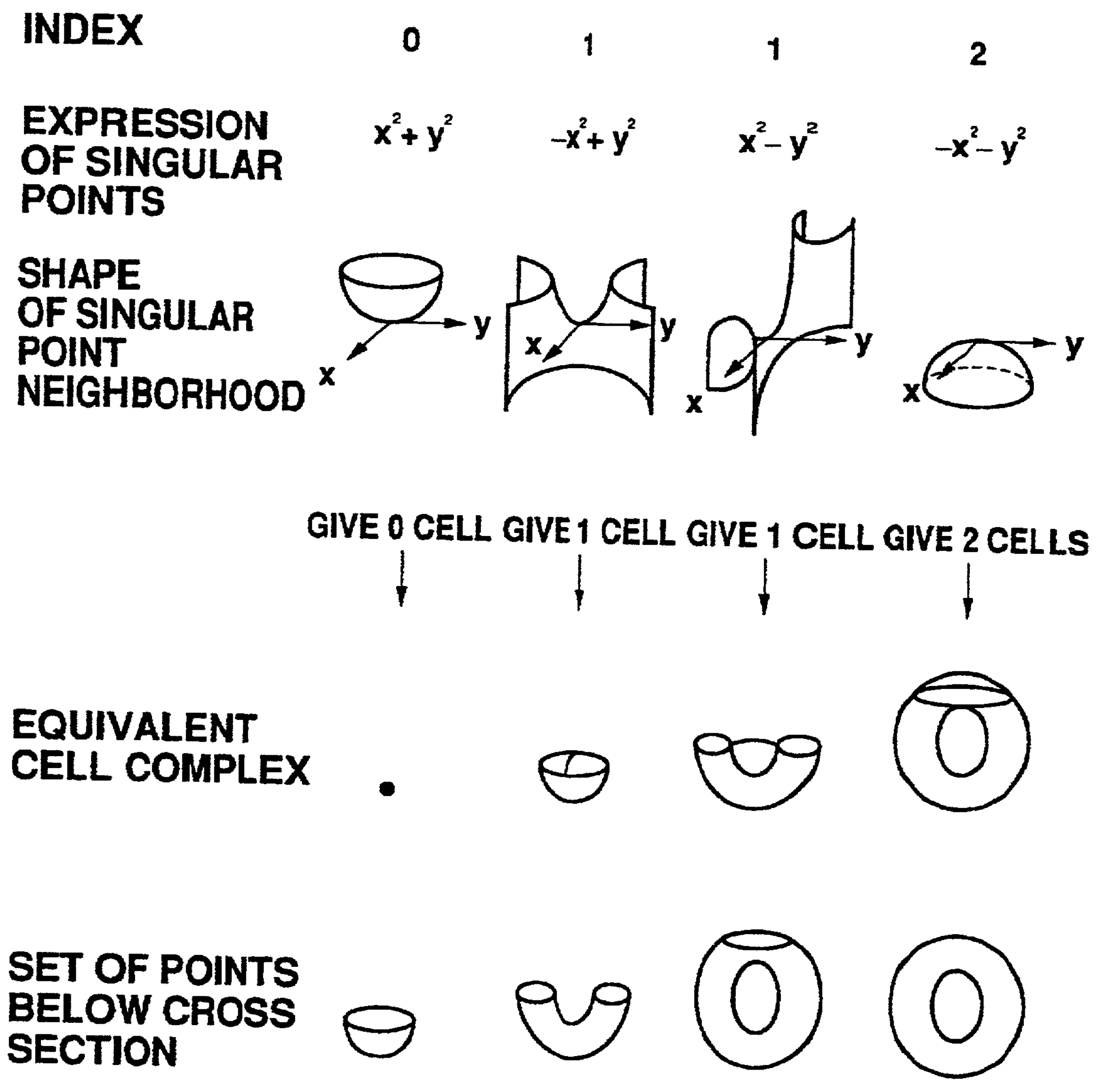

INDEX
0
1
1
2
EXPRESSION OF SINGULAR POINTS
x^2+y^2
$-x^2+y^2$
x^2-y^2
$-x^2-y^2$
SHAPE OF SINGULAR POINT NEIGHBORHOOD
x
y
GIVE 0 CELL
GIVE 1 CELL
GIVE 1 CELL
GIVE 2 CELLS
EQUIVALENT CELL COMPLEX
SET OF POINTS BELOW CROSS SECTION

FIG. 1

FIG. 2(a)

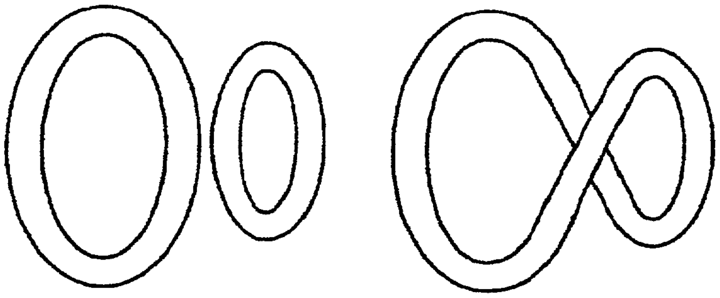


FIG. 2(b)

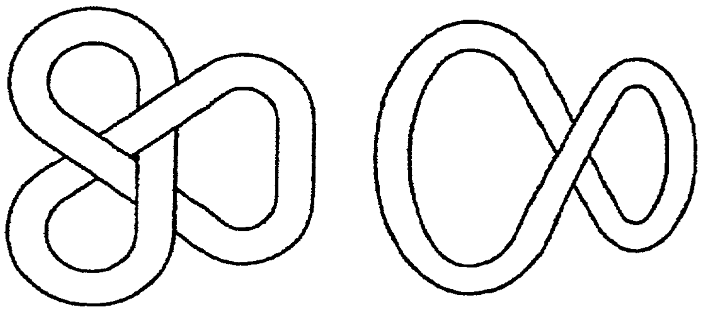


FIG. 2(c)

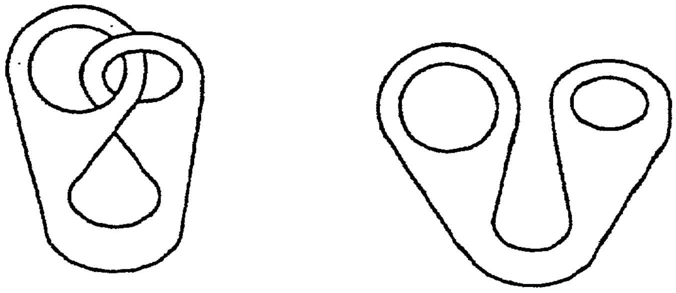


FIG. 3(a)

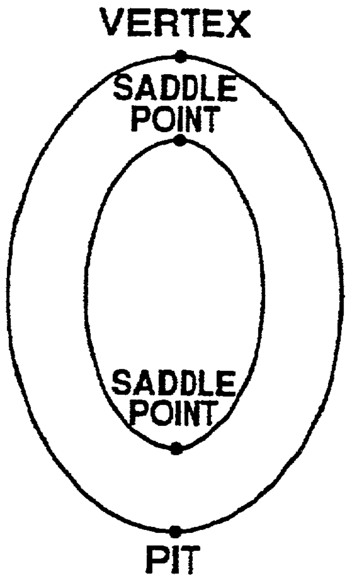


FIG. 3(b)

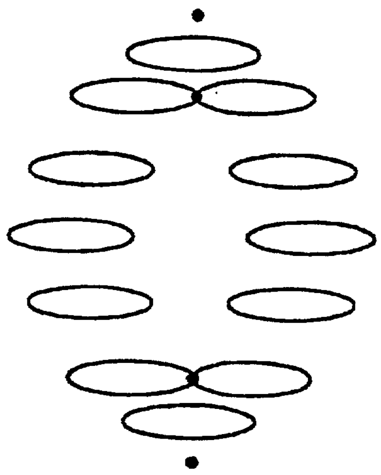


FIG. 3(c)

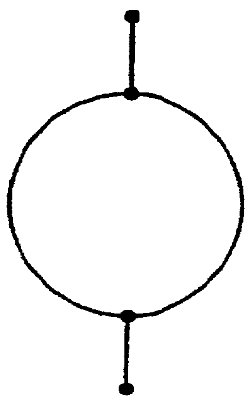


FIG. 4(a)

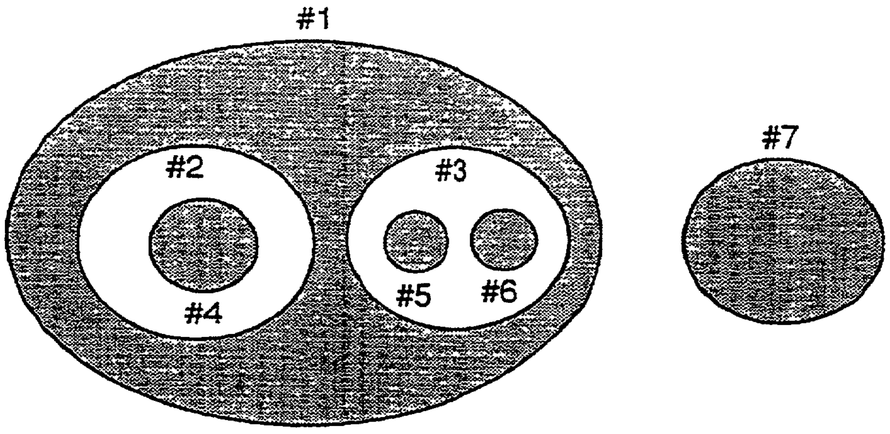
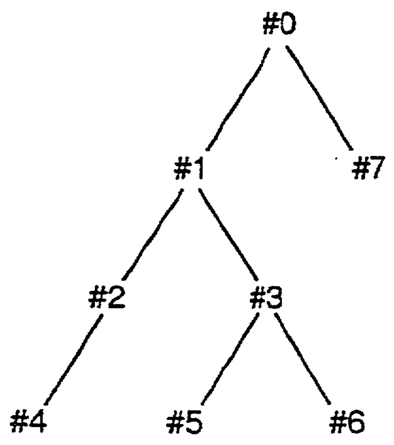


FIG. 4(b)



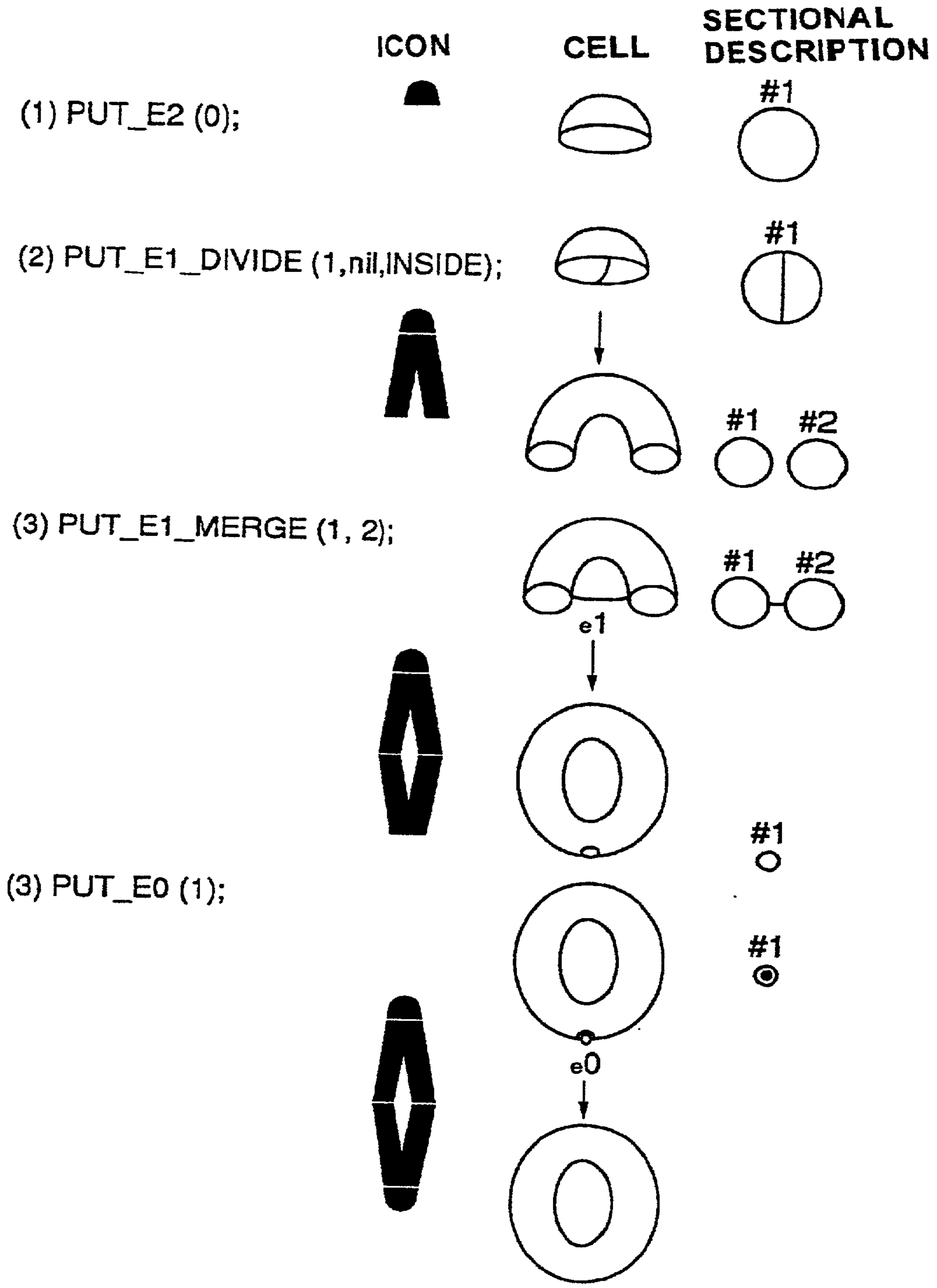

ICON
CELL
SECTIONAL DESCRIPTION
(1) PUT_E2 (0);
#1
(2) PUT_E1_DIVIDE (1,nil,INSIDE);
#1
#1 #2
(3) PUT_E1_MERGE (1, 2);
#1 #2
e1
#1
(3) PUT_E0 (1);
#1
e0

FIG. 5

```
program operators(input, output);
constant
            enabled = true;
            disabled = false;
            inside = true;
            outside = false;
            end_of_list = -1;
type
            contour_number = 0..max_contour_number;
            child_list = array[1..maxchildren] of contour_number;
            pointer_to_child_list = ↑ child_list;
var
            children: array[contour_number] of pointer_to_child_list;
            parent#: array[contour_number] of contour_number;
            number of children: array[contour_number] of integer;
            most_recently_created#: contour_number;
            contour_status: array[contour_number] of boolean;
```

FIG. 6

```
procedure add_listed_children(n:contour_number;clist:pointer_to_child_list);
            {details are omitted}
procedure remove_listed_children(n:contour_number;clist:pointer_to_child_list);
            {details are omitted}
function are_children(n:contour_number;clist:pointer_to_child_list);boolean;
            {details are omitted}
function in_list(n:contour_number;clist:pointer_to_child_list);boolean;
            {details are omitted}
function list_containing_only(n:contour_number):pointer_to_child_list;
var         n_as_list: pointer_to_child_list;
begin
            new(n_as_list);
            n_as_list ↑ [1]:= n;
            n_as_list ↑ [2]:= end_of_list;
            list_containing_only:= n_as_list;
end;
```

FIG. 7

```
a
procedure put_e2(n: contour_number);
begin
        if (contour_status[n] = disabled) then go to error;
        create_new_contour;
        add_listed_childred(n,list_containing_only(most_recently_created#));
end;
b
procedure put_e0(n: contour_number);
begin
        if ((contour_status[n] = disabled) or not all_successor_disabled(n))
              then goto error;
        contour_status[n]:= disabled;
end;
c
procedure put_e1_divide(n:contour_number); clist: pointer_to_child_list; inside:boolean);
begin
        if ((contour_status[n] = disabled) or (contour_status[parent#[n]]=disabled))
              then goto error;
        create_new_contour;
        add_listed_children(most_recently_created#, clist);
        if(not inside and are_children(parent#[n], clist)
        and not in_list(n, list)) or (clist = nil)) )
              then begin
                 remove_listed_children(parent#[n], clist);
                 add_listed_children(n,list_containing_only(most_recently_created#));
              end
        else if (inside and(are_children(n, clist) or (clist = nil)))
              then begin
                 remove_listed_children(n, clist);
                 add_listed_children(parent#[n],list_containing_only(most_recently_created#));
              end
        else go to error;
end;
d
procedure put_e1_merge(c1:contour_number; c2:contour_number);
begin
        if ((contour_status[c1] = disabled) or (contour_status[c2] = disabled))
              then goto error;
        if (c1 = parent#[c2]) then
              add_listed_children(parent#[c1], children[c2]);
        else if (parent#[c1] = parent#[c2] then
              add_listed_children(c1, children[c2]);
        else go to error;
        remove_listed_child(parent#[c2], list_containing_only(c2));
        contour_status[c2]:= disabled;
end;
```

FIG. 8

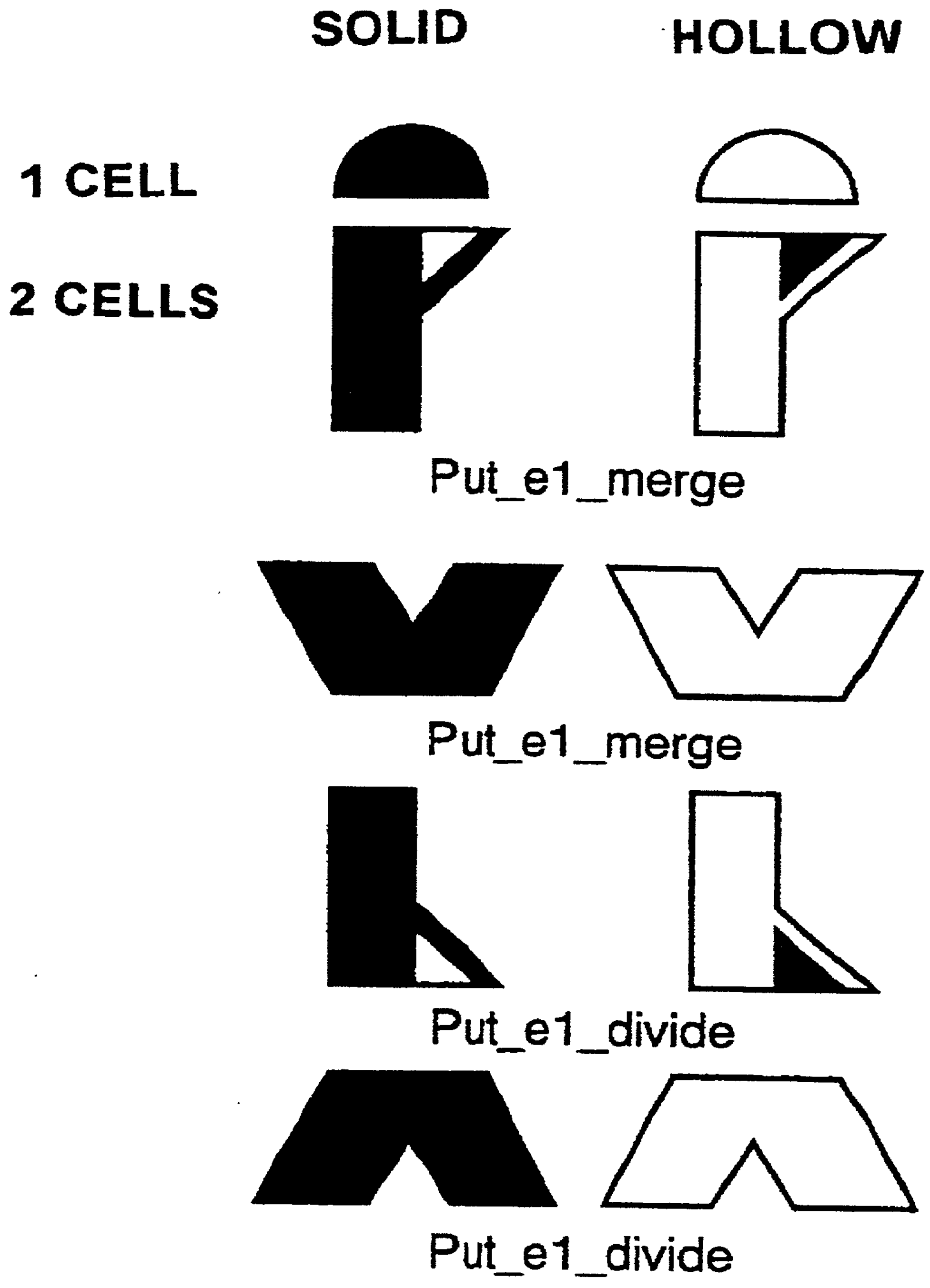

SOLID
HOLLOW
1 CELL
2 CELLS
Put_e1_merge
Put_e1_merge
Put_e1_divide
Put_e1_divide

FIG. 9

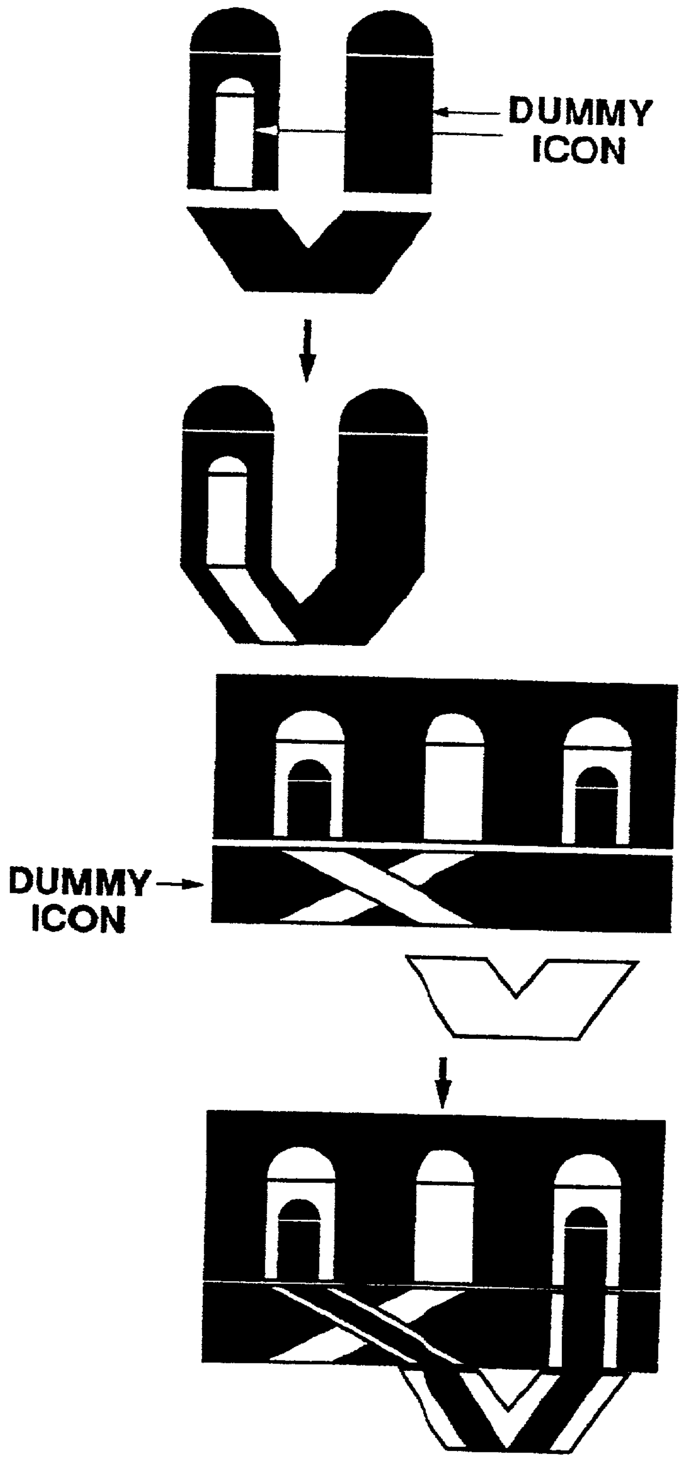


FIG. 10

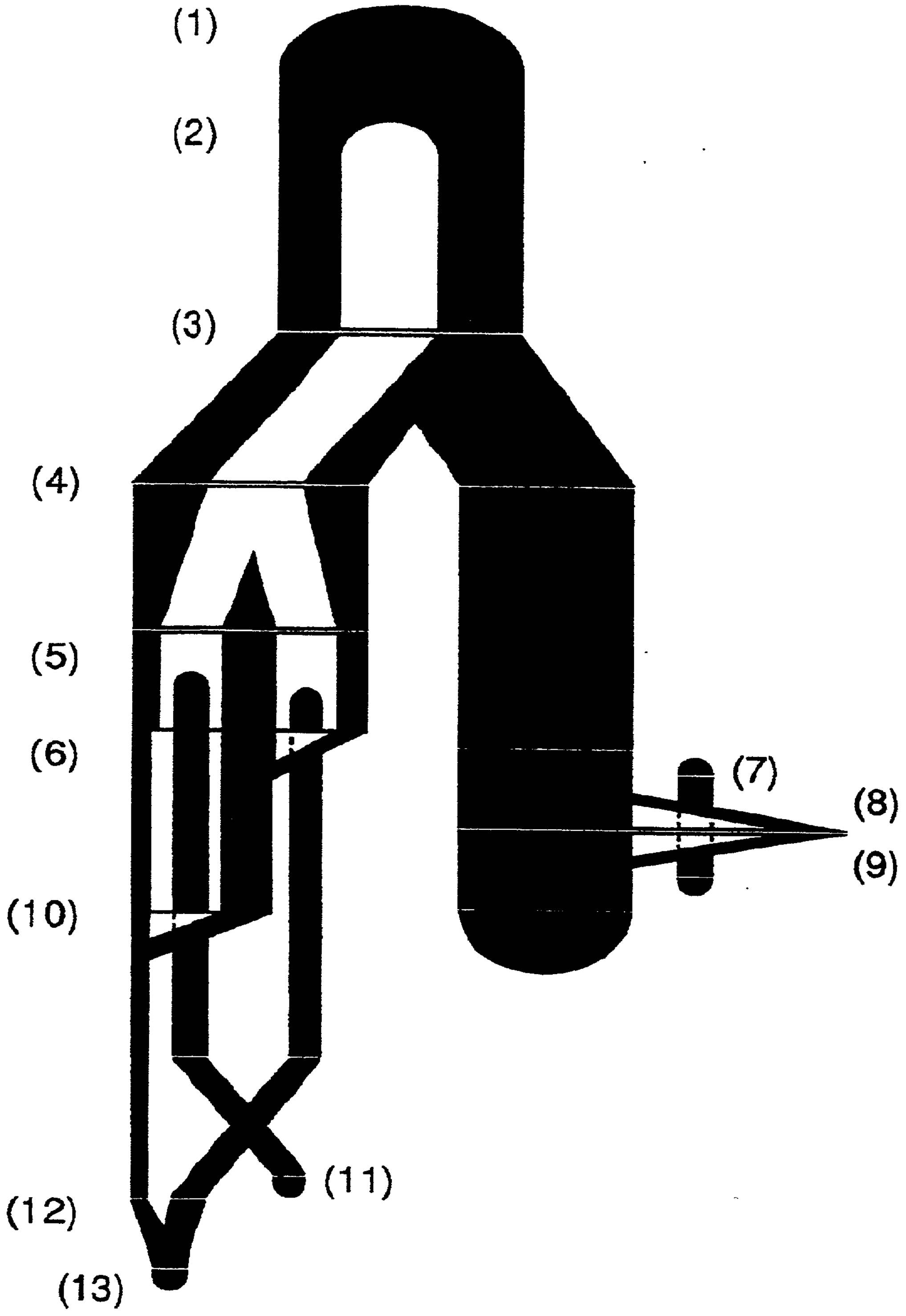


Fig. 11

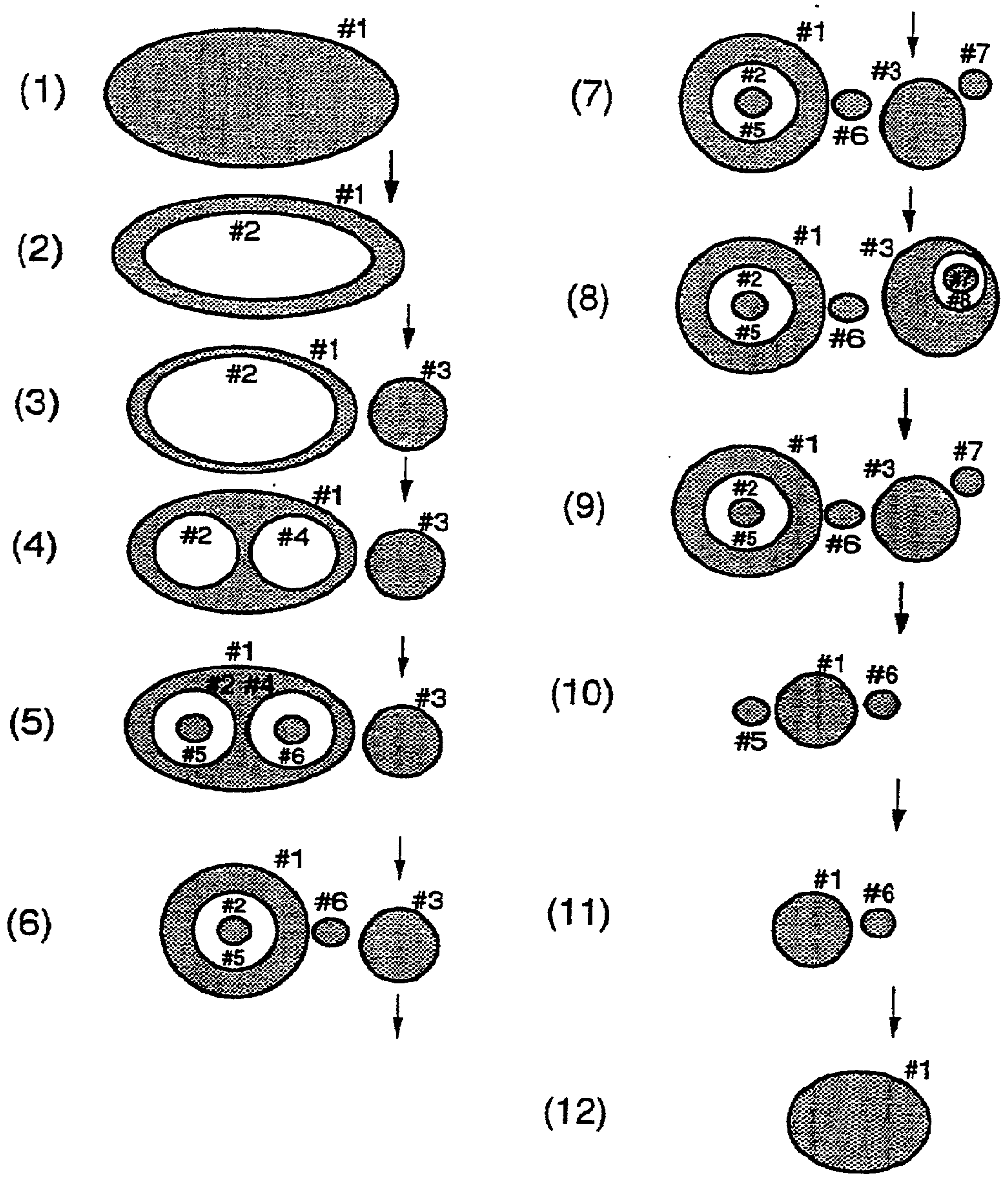

(1)
#1
(2)
#1
#2
(3)
#1
#2
#3
(4)
#1
#2
#4
#3
(5)
#1
#2
#4
#5
#6
#3
(6)
#1
#2
#5
#6
#3
(7)
#1
#2
#5
#6
#3
#7
(8)
#1
#2
#5
#6
#3
#7
#8
(9)
#1
#2
#5
#6
#3
#7
(10)
#1
#5
#6
(11)
#1
#6
(12)
#1

FIG. 12

```
 1. PUT_E2(0);
 2. PUT_E2(1);
 3. PUT_E1_DIVIDE(1, nil, INSIDE);
 4. PUT_E1_DIVIDE(2, nil, INSIDE);
 5. PUT_E2(2); PUT_E2(4);
 6. PUT_E1_MERGE(1, 4);
 7. PUT_E2(0);
 8. PUT_E1_DIVIDE(3, list_containing_only(7), OUTSIDE);
 9. PUT_E1_MERGE(3, 8); PUT_E0(7); PUT_E0(3);
10. PUT_E1_MERGE(1, 2);
11. PUT_E0(5);
12. PUT_E1_MERGE(1, 6);
13. PUT_E0(1);
```

FIG. 13

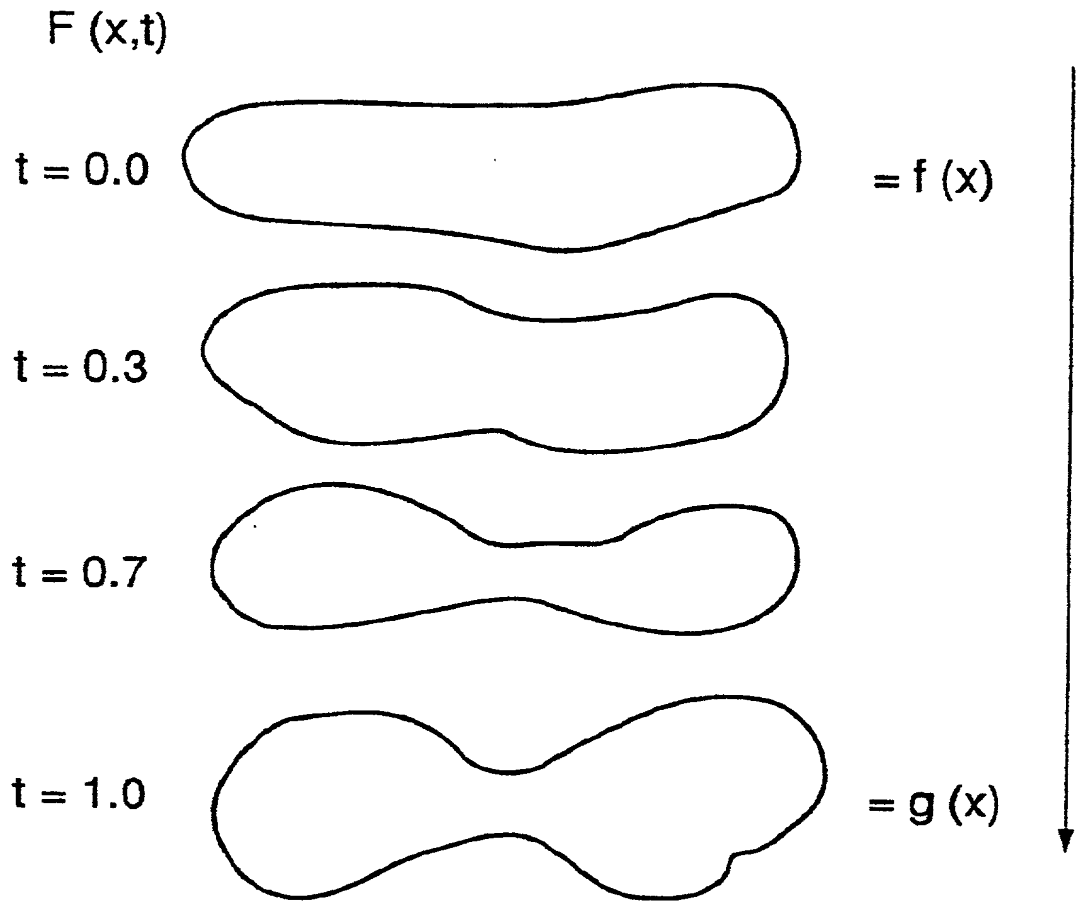


FIG. 14

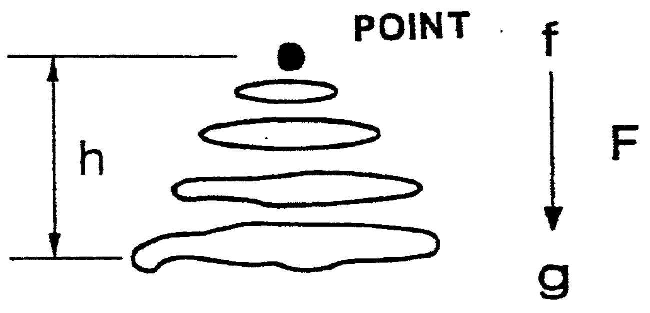


FIG. 15

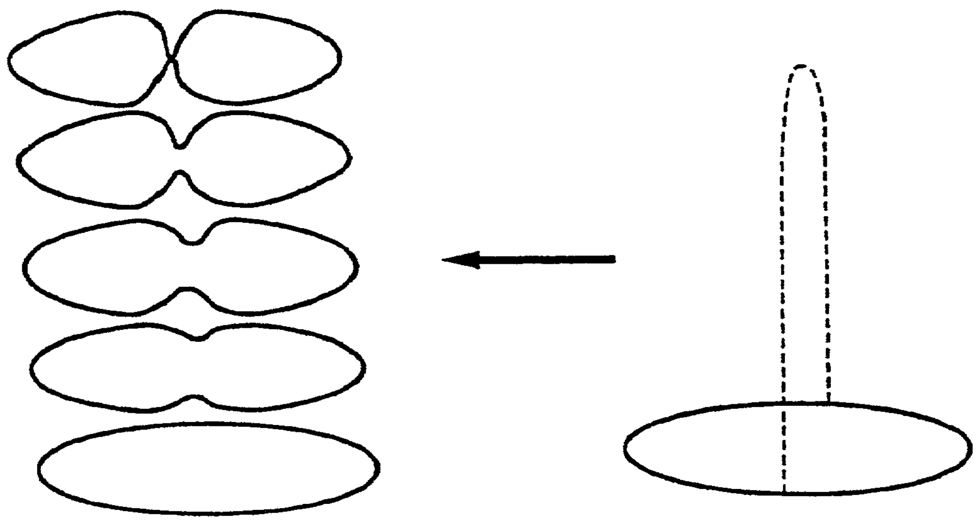


FIG. 16

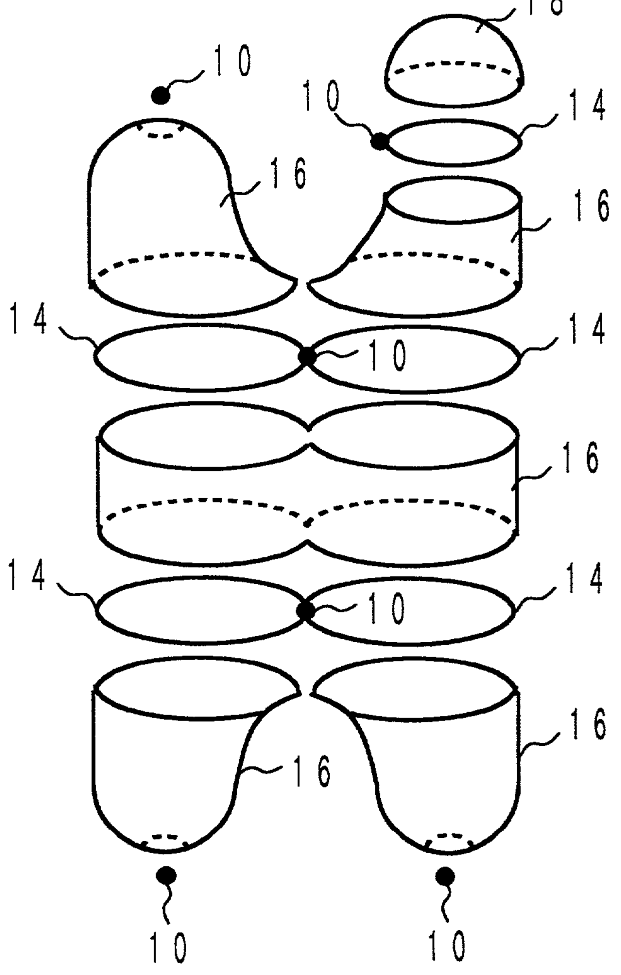


FIG. 17

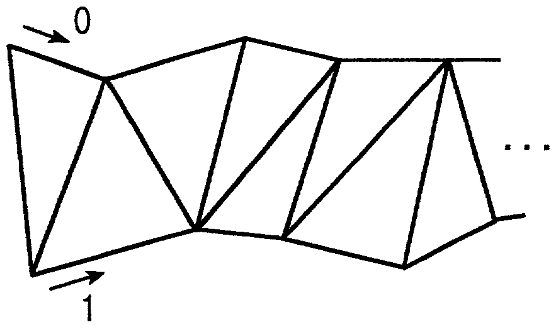


FIG.18

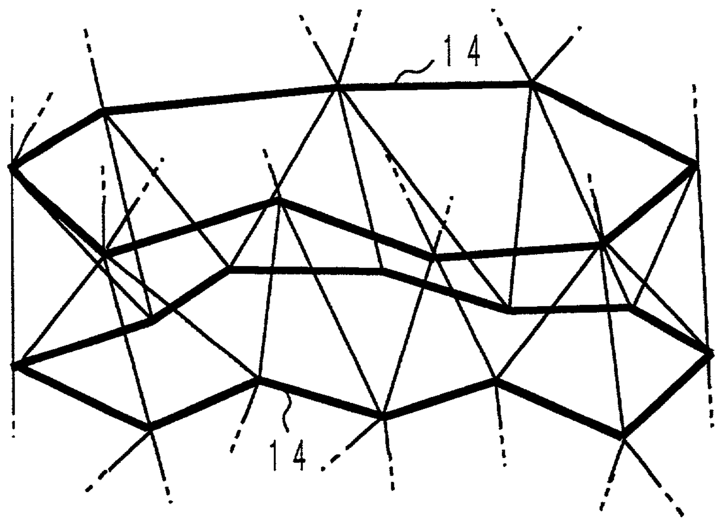


FIG.19

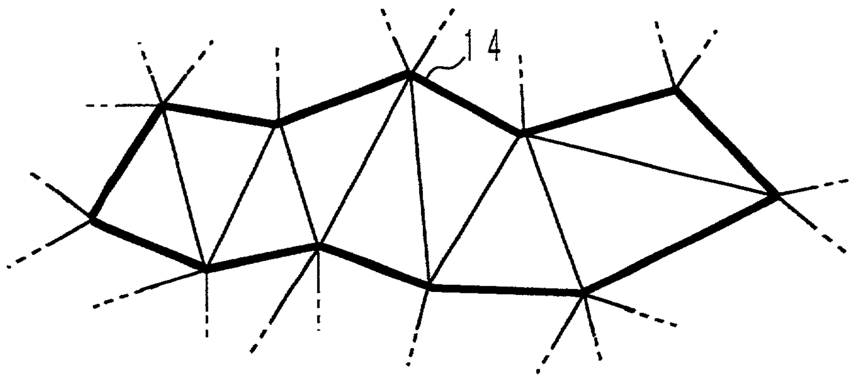


FIG.20

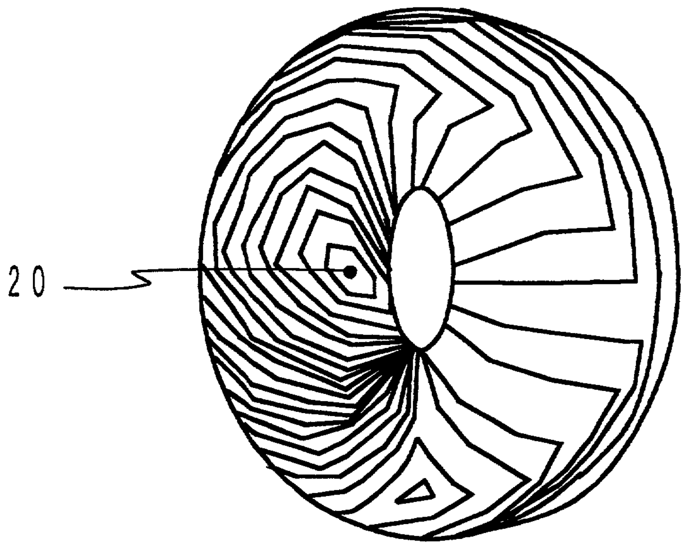


FIG.21

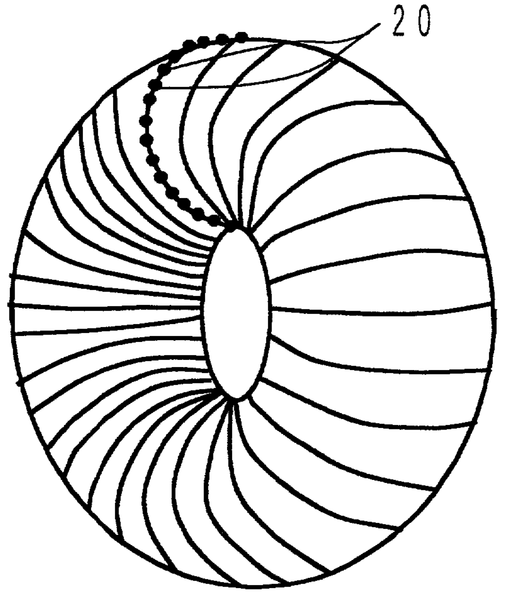


FIG.22

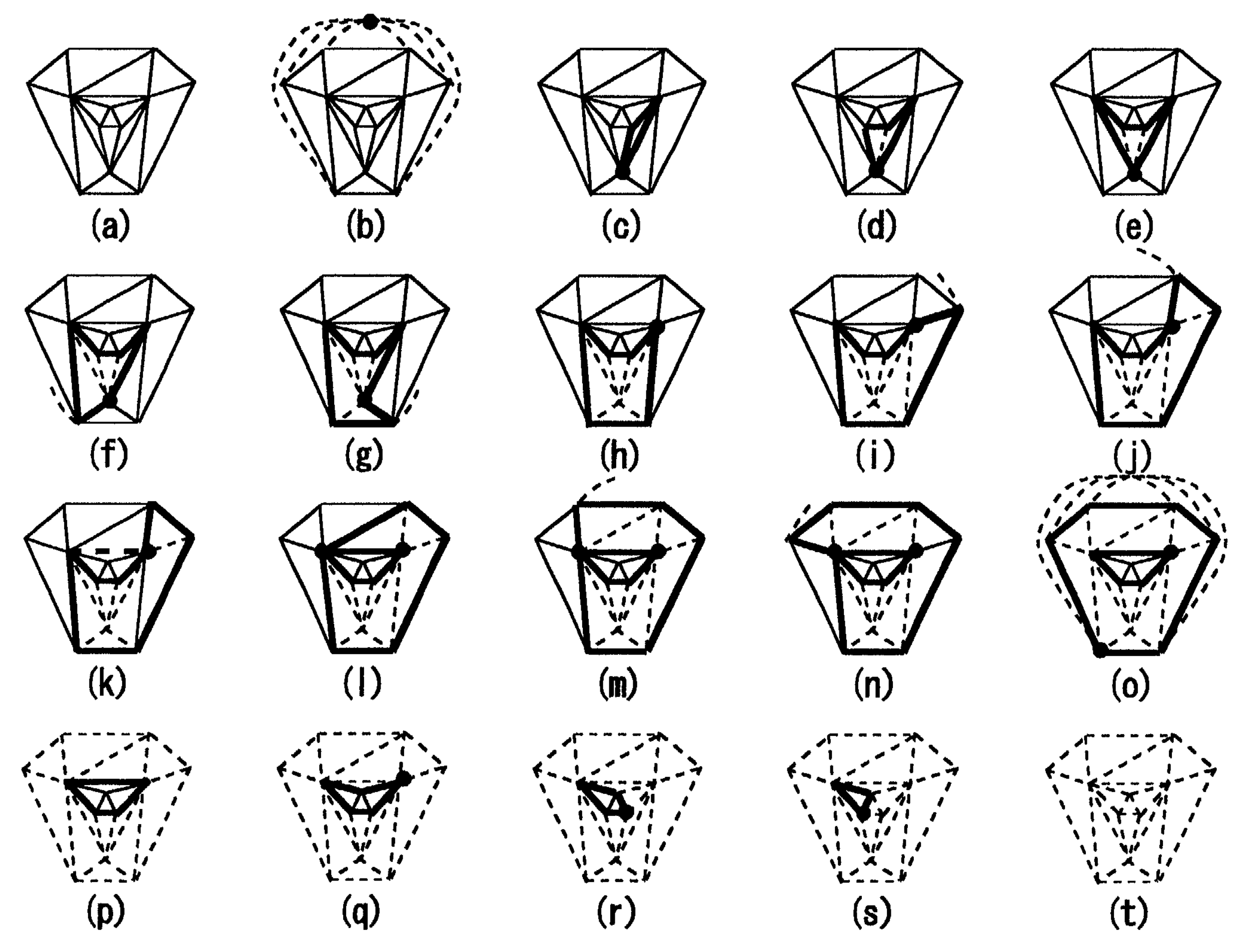


FIG.23

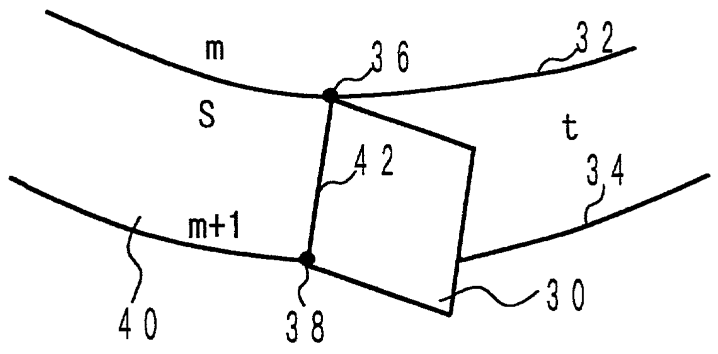


FIG.24

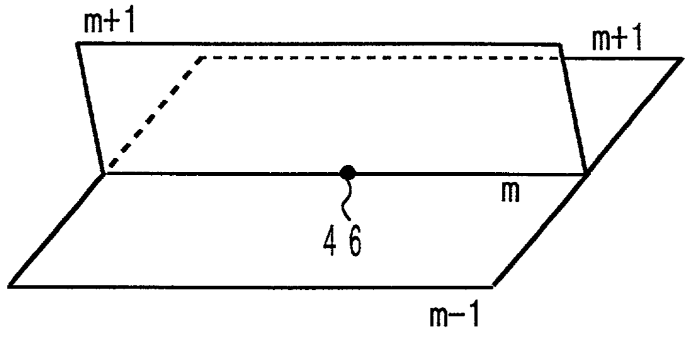
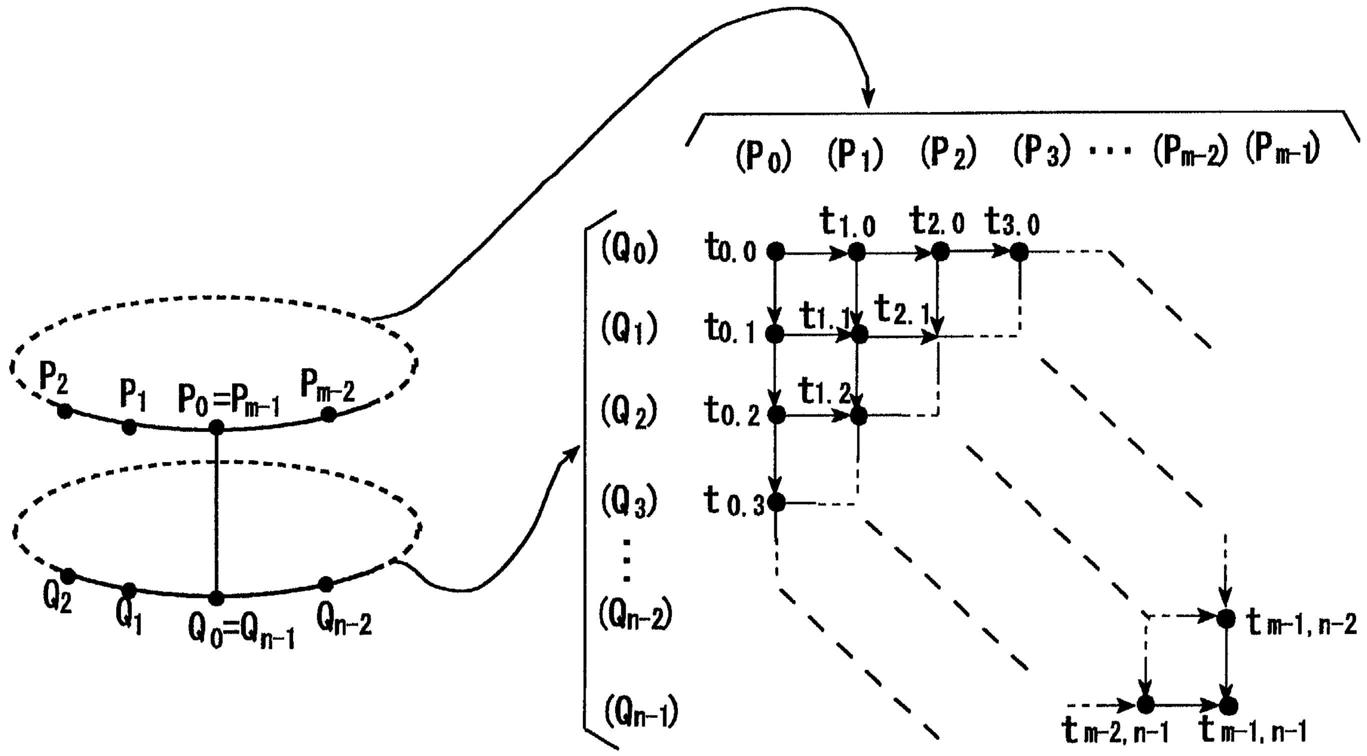


FIG.25



P_2
P_1
$P_0=P_{m-1}$
P_{m-2}
Q_2
Q_1
$Q_0=Q_{n-1}$
Q_{n-2}
(P_0) (P_1) (P_2) (P_3) $\cdots$ (P_{m-2}) (P_{m-1})
(Q_0)
(Q_1)
(Q_2)
(Q_3)
(Q_{n-2})
(Q_{n-1})
$t_{0.0}$
$t_{1.0}$
$t_{2.0}$
$t_{3.0}$
$t_{0.1}$
$t_{1.1}$
$t_{2.1}$
$t_{0.2}$
$t_{1.2}$
$t_{0.3}$
$t_{m-1,n-2}$
$t_{m-2,n-1}$
$t_{m-1,n-1}$

FIG.26

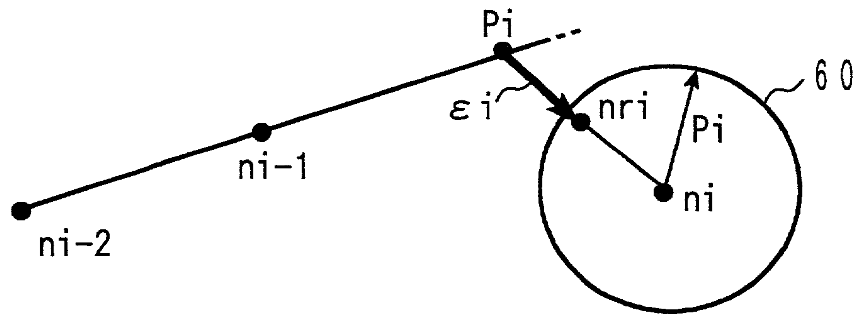


FIG.27

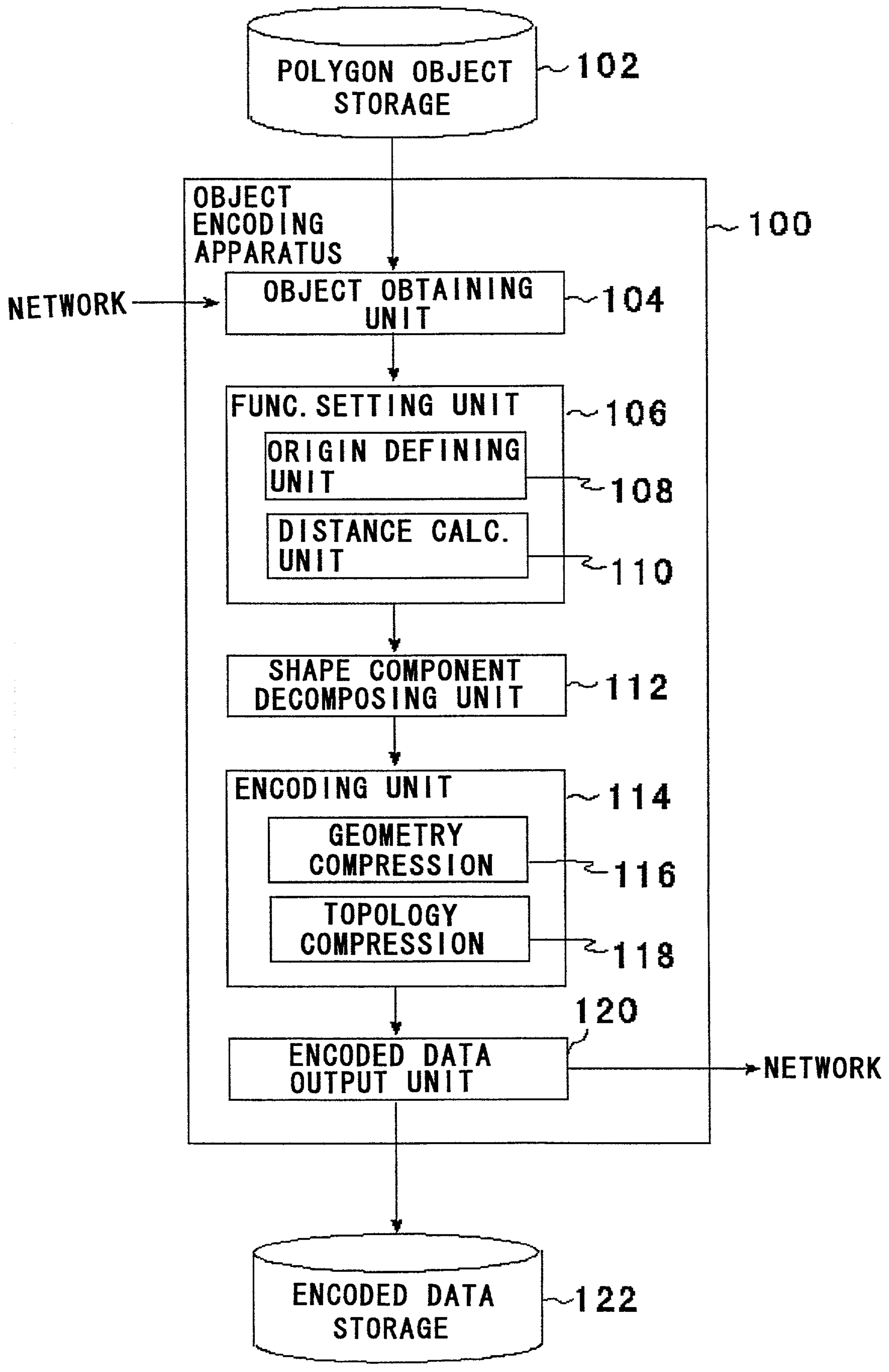

POLYGON OBJECT STORAGE
102
OBJECT ENCODING APPARATUS
100
NETWORK
OBJECT OBTAINING UNIT
104
FUNC. SETTING UNIT
106
ORIGIN DEFINING UNIT
108
DISTANCE CALC. UNIT
110
SHAPE COMPONENT DECOMPOSING UNIT
112
ENCODING UNIT
114
GEOMETRY COMPRESSION
116
TOPOLOGY COMPRESSION
118
120
ENCODED DATA OUTPUT UNIT
NETWORK
ENCODED DATA STORAGE
122

FIG.28

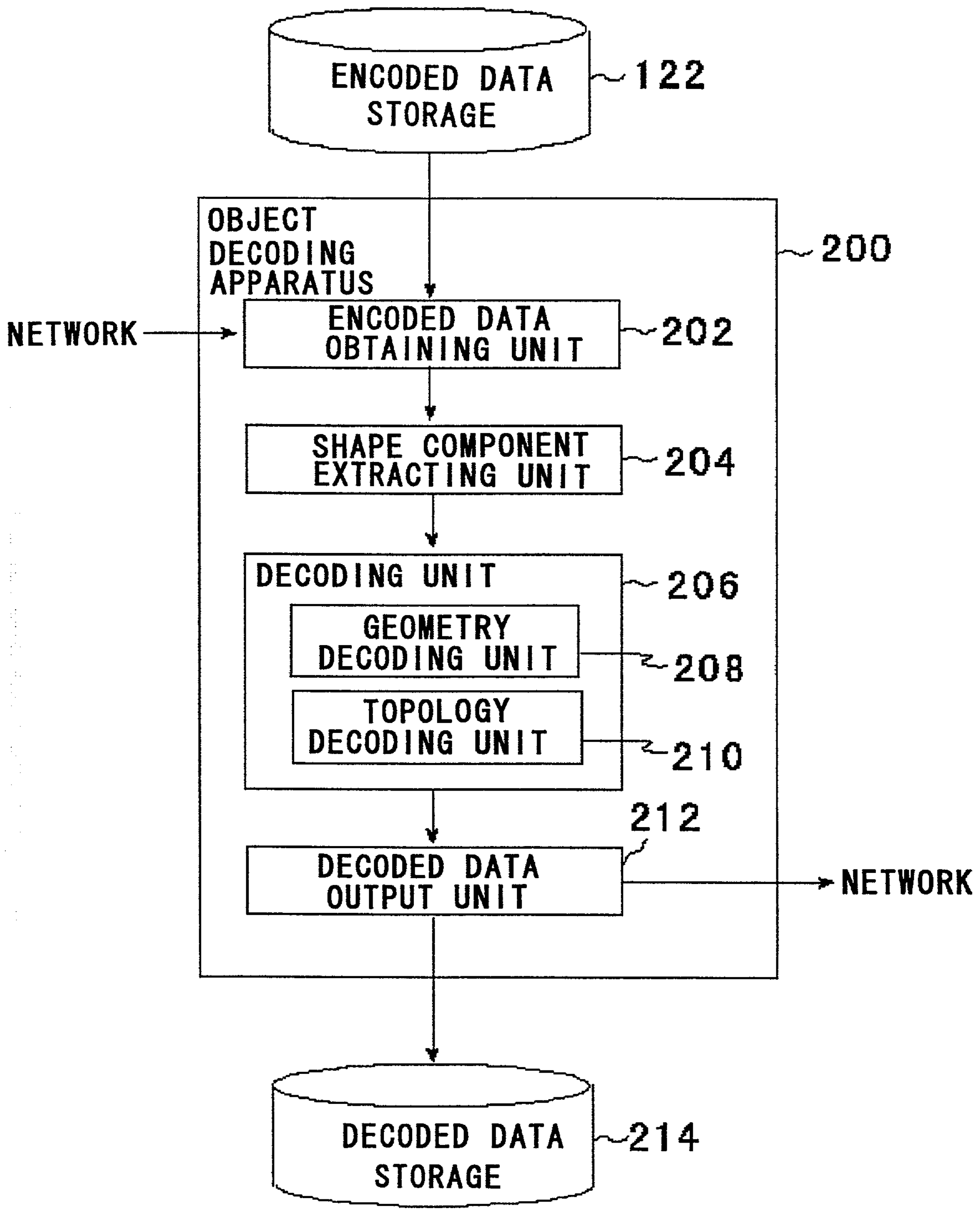

ENCODED DATA STORAGE
122
OBJECT DECODING APPARATUS
200
NETWORK
ENCODED DATA OBTAINING UNIT
202
SHAPE COMPONENT EXTRACTING UNIT
204
DECODING UNIT
206
GEOMETRY DECODING UNIT
208
TOPOLOGY DECODING UNIT
210
212
DECODED DATA OUTPUT UNIT
NETWORK
DECODED DATA STORAGE
214

FIG.29